On the Connections between Relational and XML Probabilistic Data Models

Antoine Amarilli[1] and Pierre Senellart[2]

[1] École normale supérieure, Paris, France
& Tel Aviv University, Tel Aviv, Israel
antoine.amarilli@ens.fr
http://a3nm.net/
[2] Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI, Paris, France
& The University of Hong Kong, Hong Kong
pierre.senellart@telecom-paristech.fr
http://pierre.senellart.com/

Abstract. A number of uncertain data models have been proposed, based on the notion of compact representations of probability distributions over possible worlds. In probabilistic relational models, tuples are annotated with probabilities or formulae over Boolean random variables. In probabilistic XML models, XML trees are augmented with nodes that specify probability distributions over their children. Both kinds of models have been extensively studied, with respect to their expressive power, compactness, and query efficiency, among other things. Probabilistic database systems have also been implemented, in both relational and XML settings. However, these studies have mostly been carried out independently and the translations between relational and XML models, as well as the impact for probabilistic relational databases of results about query complexity in probabilistic XML and vice versa, have not been made explicit: we detail such translations in this article, in both directions, study their impact in terms of complexity results, and present interesting open issues about the connections between relational and XML probabilistic data models.

Keywords: probabilistic data, relational data, XML

1 Introduction

A variety of systems have been put forward to represent probabilistic data and cover the needs of the various applications that produce and process uncertain data. In particular, both relational [1] and XML [2] probabilistic data models have been studied in depth, and have been investigated in terms of expressive power, query complexity, underlying algorithms, update capabilities, and so on. Similarly, systems have been developed to query probabilistic relational databases (e.g., MayBMS [3] and Trio [4]) or probabilistic documents (e.g., ProApproX [5] and [6]). By and large, these two lines of work have been conducted independently, and the results obtained have not been connected to each other.

The purpose of this article is to give a general overview of probabilistic relational and XML data models, describe the various query languages over these models, present how one can encode queries and probabilistic instances of each one of these models into the other, and investigate the consequences of these encodings in terms of complexity results. We focus specifically on the existence and efficiency of translations across models, and on the transposition of query complexity results, rather than on a systems aspect. Section 2 introduces the probabilistic representation systems we consider, and Section 3 the corresponding query languages. We describe encodings of relations into XML in Section 4, and of XML into relations in Section 5.

2 Data Models

Probabilistic data models are a way to represent a probability distribution over a set of *possible worlds* that correspond to possible states of the data. We focus on the *discrete* and *finite* case where the set of possible worlds is finite; each possible world is associated with a probability value, i.e., a rational in $(0, 1]$, such that the sum of probabilities of all possible worlds is 1. States of the data that are not part of the set of possible worlds have a probability of 0.

A straightforward probabilistic data model is to materialize explicitly the collection of possible worlds with their probability. However, this straightforward representation is not *compact*; it is as large as the number of possible worlds, and any operation on it (such as answering a query) must iterate over all possible worlds. For this reason, probabilistic data models usually represent the set of possible worlds and the probability distribution in an implicit fashion.

Probabilistic data models usually achieve a trade-off between expressiveness and computational complexity: ability to represent as many different kinds of distributions as possible on the one hand, tractability of various operations on the model, such as querying, on the other hand. In this section, we present probabilistic data models for relational data and for XML data: in both settings, we will move from the less expressive to the more expressive.

2.1 Relational models

Probabilistic models for relational data have usually been built on top of models for representing *incomplete* information. Incomplete information defines a set of possible worlds (the possible completions of the existing information), and probabilistic models usually add some probability scores for each of the possible worlds. See [1] for a general survey of probabilistic relational models.

The tuple-independent model. One of the simplest ideas to define a probabilistic relational model is the *tuple-independent model* [7,8] (also known as tables with *maybe tuples* [9] or *probabilistic ?-tables* [10]). In this model, a probabilistic database is an ordinary database where tuples carry a probability of actually occurring in the database, independently from any other tuple. Formally, given

a relational schema Σ, an instance $\widehat{D}$ of the probabilistic relational schema $\widehat{\Sigma}$ is defined as a Σ-instance in which every tuple $R(\mathbf{a})$ is annotated with a rational probability value $\Pr_{\widehat{D}}(R(\mathbf{a})) \in (0, 1]$ (with tuples absent from $\widehat{D}$ having probability 0). The probability of a Σ-instance D according to $\widehat{D}$ is then defined as $\Pr_{\widehat{D}}(D) = \prod_{R(\mathbf{a})\in D} \Pr_{\widehat{D}}(R(\mathbf{a})) \prod_{R(\mathbf{a})\notin D} \big(1 - \Pr_{\widehat{D}}(R(\mathbf{a}))\big)$, the product of the probabilities in $\widehat{D}$ of retaining the tuples occurring in D and dropping the others (note that the second product is infinite but has finite support). Since each tuple is either retained or dropped, there are $2^{|\widehat{D}|}$ possible worlds of non-zero probability, and we can check that their probabilities sum to 1.

This model is simple but not very expressive because of the independence assumption. As an example, if the original schema has a predicate $R(A, B)$ with a key constraint $A \to B$, we cannot give non-zero probability to instances $\{R(a, b)\}$ and $\{R(a, b')\}$ without giving non-zero probability to instance $\{R(a, b), R(a, b')\}$ that violates the key constraint.

The block-independent-disjoint model. An extension of the tuple-independent model to support simple mutually exclusive choices is the *block-independent disjoint* model [11,12]. In this variant, we assume that every predicate R of Σ is of the form $R(\mathbf{K}, \mathbf{A})$, where the attributes have been partitioned into two sets: $\mathbf{K}$, the *possible worlds key*, and $\mathbf{A}$, the *value attribute set*. Besides, we require that the key constraint $\mathbf{K} \to \mathbf{A}$ holds. The BID schema $\widehat{\Sigma}$ is defined as before with the added constraint that $\sum_{\mathbf{a}\in\mathbf{A}} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a})) \leqslant 1$ for every predicate $\widehat{R}(\mathbf{K}, \mathbf{A})$ and possible worlds key $\mathbf{k} \in \mathbf{K}$. Intuitively, for each $\mathbf{k}$, there is a probability distribution on the possible *exclusive* choices of $\mathbf{a}$ (including the default option of choosing no $\mathbf{a}$). A $\widehat{\Sigma}$-instance $\widehat{D}$ defines the probability distribution $\Pr_{\widehat{D}}(D) = \prod_{R(\mathbf{k},\mathbf{a})\in D} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a})) \prod_{R(\mathbf{k},\bullet)\notin D} \Big(1 - \sum_{\widehat{R}(\mathbf{k},\mathbf{a},p)\in\widehat{D}} \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a}))\Big)$ with the added constraint that there are no duplicate facts $R(\mathbf{k}, \mathbf{a})$ and $R(\mathbf{k}, \mathbf{a}')$ for any $R(\mathbf{K}, \mathbf{A}) \in \widehat{\Sigma}$, $\mathbf{K} \in \mathbf{K}$, $\mathbf{a}, \mathbf{a}' \in \mathbf{A}$, $\mathbf{a} \neq \mathbf{a}'$ (otherwise the probability is 0).

An example BID database, consisting of a single Customer($\mathbf{Id}$, Name, City) relation (where $\mathbf{Id}$ is the key) is given in Fig. 1. Mutually exclusive names and cities are given for the two customers, with the possibility also that neither customer exists in a possible world. Such an uncertain table may be obtained, for instance, following a data integration process from conflicting sources.

Intuitively, instances are drawn from a BID instance by picking one of the mutually exclusive choices of $\mathbf{a}$ within each *block* defined by a choice of $\mathbf{k}$, and doing so independently across the blocks. The BID model is more expressive than the tuple-independent model (which can be seen as the case in which all attributes of every predicate are taken as the possible worlds key $\mathbf{K}$).

Of course, the structure of the BID model is still unable to represent many kinds of probability distributions. For instance, if instance $\{R(a, b), R(a', b')\}$ has non-zero probability in a relation whose possible worlds key is the first attribute, then instances $\{R(a, b)\}$ and $\{R(a', b')\}$ will also have non-zero probability.

Customer			
Id	Name	City	Pr
1	John	New York	0.4
1	Johny	New York	0.2
1	John	Boston	0.1
2	Mary	Boston	0.4
2	Maria	Boston	0.1

Fig. 1. Example BID database

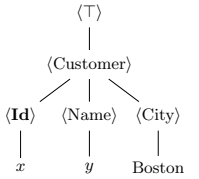

Fig. 2. Example of a TPQJ to encode Customer(x, y, Boston)

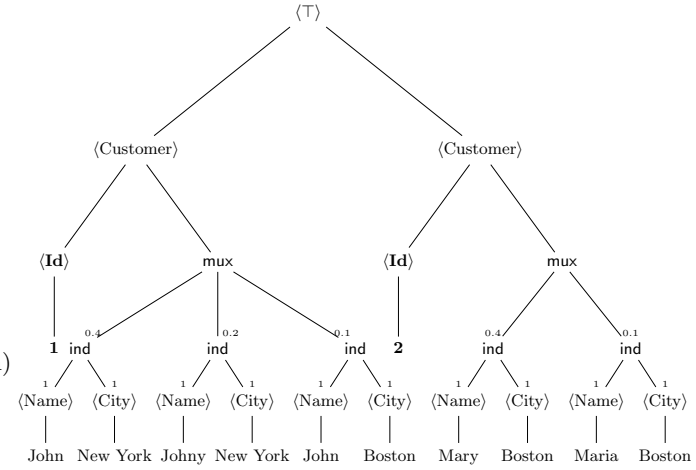

Fig. 3. Example $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ tree encoding the BID database of Fig. 1

The pc-tables model. We will now present a much more expressive model: *probabilistic c-tables* (a.k.a. pc-tables) [9,3]. To simplify the writing, we will restrict our exposition to *Boolean pc-tables*, that are known to be as expressive as general pc-tables [9].

Given a relational schema Σ, an instance $\widehat{D}$ of the probabilistic relational schema $\widehat{\Sigma}$ is a Σ-instance which annotates every tuple $R(\mathbf{a})$ with a Boolean formula $\mathrm{Con}_{\widehat{D}}(R(\mathbf{a}))$ over a global finite set of *variables* $\mathcal{V}$ and provides an additional relation B assigning a probability value $\Pr_{\widehat{D}}(x) \in (0, 1]$ to each variable $x \in \mathcal{V}$ occurring in the instance (intuitively, the probability that x is true). Given an assignment ν from the set $\overline{\mathcal{V}}$ of Boolean assignments (i.e., functions from $\mathcal{V}$ to Boolean values $\{\mathfrak{t}, \mathfrak{f}\}$), we define D_ν to be the Σ-instance obtained from $\widehat{D}$ by removing each fact $R(\mathbf{a})$ such that $\mathrm{Con}_{\widehat{D}}(R(\mathbf{a}))$ evaluates to $\mathfrak{f}$ under ν. We then define the probability of assignment ν as: $\Pr_{\widehat{D}}(\nu) = \left(\prod_{x \in \mathcal{V} \text{ s.t. } \nu(x)=\mathfrak{t}} \Pr_{\widehat{D}}(x)\right)\left(\prod_{x \in \mathcal{V} \text{ s.t. } \nu(x)=\mathfrak{f}} \left(1 - \Pr_{\widehat{D}}(x)\right)\right)$. The probability of a Σ-instance D is then: $\Pr_{\widehat{D}}(D) = \sum_{\nu \in \overline{\mathcal{V}} \text{ s.t. } D_\nu = D} \Pr_{\widehat{D}}(\nu)$. Intuitively, Σ-instances are obtained by drawing an assignment of the variables independently according to B and keeping the facts where the condition is true.

It is not very hard to see that *any* probability distribution over *any* finite set of possible worlds can be modeled with a pc-table, given sufficiently many tuples and variables. In particular, BID tables can be expressed as pc-tables, and a direct polynomial-time translation is straightforward.

2.2 XML Models

We will now present probabilistic models for XML data. This presentation is inspired by [2] in which more details can be found. Given an infinite set of labels $\mathcal{L}$, an XML document is a rooted, unordered, directed, and finite tree

where each node has a *label* in $\mathcal{L}$. For simplicity, we will always require that the root of an XML document (probabilistic or not) has a fixed label $\sigma_{\mathrm{r}} \in \mathcal{L}$. We denote by $\mathcal{X}$ the collection of XML documents over $\mathcal{L}$.

The $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ *data model.* We define a probabilistic document in the $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ model as an XML document over $\mathcal{L} \cup \{\mathsf{ind}\}$ (ind for *independent*), where the outgoing edges of ind nodes carry a rational number in $(0, 1]$ as label. Such documents will define a probability distribution over $\mathcal{X}$ that we will describe as a sampling process: in a top-down fashion, for every ind node x, independently choose to keep or discard each of its children according to the probability label on each outgoing edge, and, in the father y of x, replace x by the descendants of x that were chosen, removing x and its unchosen descendants. Perform this process independently for every node ind.

Once all ind nodes have been processed and removed in this way, the outcome is ar XML document over $\mathcal{L}$, and its probability is the conjunction of the independent events of all the results of the draws at ind nodes. The probability of a document is the sum of the probability of all outcomes leading to it. Note that multiple different outcomes may lead to the same document.

The $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ *data model.* In the same fashion, we can define the $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ data model (mux for *mutually exclusive*), in which we also require that the outgoing edges of each mux node x carry a rational number in $(0, 1]$ as label such that the labels of all the outgoing edges of x sum to at most 1. The sampling process proceeds as described above, except that each mux node choses at most one of its children according to the probability label on each outgoing edge.

Of course, we can define the $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ data model as the data model in which both ind and mux nodes are allowed, which was studied under the name ProTDB in [13]. An example $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ is given in Fig. 3; as we shall see in Section 4, it is an encoding of the BID database of Fig. 1.

The $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *data model.* Finally, we define the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ data model (fie for *formula of independent events*), in which the outgoing edges of fie nodes are labeled with Boolean formulae on some finite set $\mathcal{V}$ of Boolean variables and in which we are given a rational probability value $P(x)$ in $(0, 1]$ for each variable $x \in \mathcal{V}$. The sampling process is to draw independently the truth value of each $x \in \mathcal{V}$ according to the probability $P(x)$, and replace each fie node by its children for which the Boolean formula appearing as an edge label evaluates to t under the assignment that was drawn.

Expressive power and compactness. $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ and $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ are incomparable in terms of expressive power [14]: some probability distributions can be expressed by one and not by the other. Thus, $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ is strictly more expressive than these two, and it is easy to see that one can actually use it to represent *any* finite probability distribution over $\mathcal{X}$ (recall that the root label of all possible documents is fixed).

Any $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic document can be transformed in polynomial time into an $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ document (where all Boolean formulae are conjunctions) which yields the same probability distribution: ind nodes can be encoded with fie nodes by creating one Boolean variable for each of their descendants; a similar encoding exists for mux nodes by first encoding n-ary mux nodes in a subtree of binary mux nodes, and then replacing each binary mux node by one fie node whose children are labeled by x and $\neg x$, where x is a fresh Boolean variable with adequate probability.

On the other hand, no polynomial time translation exists in the other direction [14], even when considering $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ probabilistic documents with conjunctions only. In other words, though $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ and $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ have the same expressive power, the latter can be exponentially more compact than the former.

3 Querying Probabilistic Data

Up to this point, we have described probabilistic data models that serve as a concise representation of a probability distribution on a set of possible worlds. However, the main interest of such models is to use them to evaluate queries on all possible worlds simultaneously and return the aggregate results as a probability distribution on the possible answers. For clarity of the exposition, we restrict ourselves to Boolean queries, that are either true or false on a given possible world. Extensions to non-Boolean queries are straightforward.

More formally, given some probabilistic data model, given a probabilistic instance $\widehat{M}$ defining a probability distribution over possible worlds $\mathbf{X}$, and given a Boolean query q that can be evaluated on each $X \in \mathbf{X}$ to produce either $\mathfrak{t}$ or $\mathfrak{f}$, we define the *probability of q on $\widehat{M}$* as: $\hat{q}(\widehat{M}) = \sum_{X \in \mathbf{X} \text{ s.t. } q(X)=\mathfrak{t}} \Pr_{\widehat{M}}(X)$. Evaluating $\hat{q}$ on $\widehat{M}$ means computing the probability of q on $\widehat{M}$. This is called the *possible-worlds query semantics*.

In this section, we present known results about the complexity of query evaluation under this semantics. The query is always assumed to be fixed, i.e., we discuss the *data complexity* of query evaluation.

We will need some basic notions about complexity classes for computation and counting problems [15]. The class FP is that of computation problems solvable in deterministic polynomial time, while #P problems are those that can be expressed as the number of accepting runs of a polynomial-time nondeterministic Turing machine. A computation problem is in $\mathsf{FP}^{\#\mathsf{P}}$ if it can be solved in deterministic polynomial time with access to a #P oracle. A problem is $\mathsf{FP}^{\#\mathsf{P}}$-hard if there is a polynomial-time Turing reduction from any $\mathsf{FP}^{\#\mathsf{P}}$ problem to it.

3.1 Relational models

Typical query languages on relational data include *conjunctive queries* (CQs, i.e., select-project-joins), *unions of conjunctive queries* (disjunctions of conjunctive queries, a.k.a. UCQs), and the *relational calculus*. A CQ is *hierarchical* if for any

two variables x and y, either the intersection of the set of atoms that contain x with that of y is empty, or one of them is contained in the other (this notion can be extended to arbitrary relational calculus queries, see [1]). A CQ is *without self-join* if all atoms bear distinct relation names.

For instance, a simple CQ for the database of Fig. 1, testing whether any customer is located in Boston, is $q_{\text{Boston}} = \exists x \exists y\, \text{Customer}(x, y, \text{Boston})$. This query, having a single atom, is trivially hierarchical and without self-joins. It is easy to see that $\hat{q}_{\text{Boston}}(\hat{D}) = 0.55$, where $\widehat{D}$ is the database of Fig. 1.

Extensive results exist about the complexity of evaluating the probability of a query for these various query languages over the tuple-independent, BID, and pc-tables models. We refer to [1] for a detailed overview and summarize here the main results:

- Query evaluation for relational calculus queries over pc-tables is $\mathsf{FP}^{\#\mathsf{P}}$ [16].
- CQs of only one atom are already $\mathsf{FP}^{\#\mathsf{P}}$-hard over pc-tables, even when the Boolean formulae in the pc-table are restricted to conjunctions [17].
- For UCQs over tuple-independent databases, there is a dichotomy between $\mathsf{FP}^{\#\mathsf{P}}$-hard queries and FP queries [18]; however, the only known algorithm to determine the complexity of a query is doubly exponential in the query size, and the exact complexity of this problem is unknown.
- A similar dichotomy, but with a polynomial-time test, holds for CQs without self-joins over BIDs. [16]
- A CQ without self-joins is $\mathsf{FP}^{\#\mathsf{P}}$-hard over tuple-independent databases if and only if it is not hierarchical. [8] Being non-hierarchical is a sufficient condition for any relational calculus query to be $\mathsf{FP}^{\#\mathsf{P}}$-hard.

3.2 XML models

Tree-pattern queries with joins. The first query language that we will consider on XML data are *tree-pattern queries with joins* (TPQJs). A TPQJ q is a rooted, unordered, directed and finite tree whose nodes are labeled either with elements of $\mathcal{L}$ or with *variable symbols* taken from some infinite fixed set of variables $\mathcal{V}$, and whose edges are either *child* or *descendant* edges. Given an *assignment* $\nu : \mathcal{V} \to \mathcal{L}$, we define the *application* of ν to q (written $q[\nu]$) as the tree where each label $x \in \mathcal{V}$ is replaced by $\nu(x)$. A *match* of q in an XML document d is an assignment ν and a mapping μ from the nodes of $q[\nu]$ to the nodes of d such that:

1. For any child edge $x \to y$ in $q[\nu]$, $\mu(y)$ is a child of $\mu(x)$ in d.
2. For any descendant edge $x \to y$ in $q[\nu]$, $\mu(y)$ is a descendant of $\mu(x)$ in d.
3. For any node x of $q[\nu]$, its label is the same as that of $\mu(x)$ in d.

Intuitively, we match the query with some part of the document so that child, descendant, and fixed label constraints are respected, and all the occurrences of a variable are mapped to nodes with the same label. Tree-pattern queries (TPQs) are TPQJs where all variable symbols are distinct.

We show in Fig. 2 an example TPQJ. Here, x and y are variables, all other labels are from $\mathcal{L}$. We can show that this query, when evaluated on the $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ tree of Fig. 3, yields a probability of 0.55.

Monadic second-order logic with joins. A more expressive query language for XML documents is *monadic second-order tree logic with value joins* (MSOJ). Remember that a monadic second-order formula is a first-order formula extended with existential and universal quantification over node sets. An MSOJ query is a monadic second-order formula over the predicates $x \to y$ (y is a child of x), $\lambda(x)$ (x has label λ), and $x \sim y$ (x and y have same label, a.k.a. *value join*).

A MSO query is an MSOJ query without any value join predicate.

Query complexity. We refer to [2] for a survey of query complexity in probabilistic XML and summarize here the main results:

- Query evaluation for TPQJs over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ is $\mathsf{FP}^{\#\mathsf{P}}$ [19].
- For MSO queries over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$, query evaluation is linear-time [20].
- Any non-trivial TPQ is $\mathsf{FP}^{\#\mathsf{P}}$-hard over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$, even when the Boolean formulae of the document are restricted to conjunctions [21].
- If a TPQJ has a single join (i.e., a single variable appears twice in the query), then the following dichotomy holds over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ [19]: if it is equivalent to a join-free query, it is linear-time; otherwise it is $\mathsf{FP}^{\#\mathsf{P}}$-hard. Testing for this equivalence is Σ_2^{P}-complete. It is open whether this still holds for TPQJs with more than one join.

Note that this list of results has some similarity with that in the relational setting: a broad $\mathsf{FP}^{\#\mathsf{P}}$ membership result, hardness of models with Boolean formulae, even with just conjunctions, and a dichotomy between $\mathsf{FP}^{\#\mathsf{P}}$-hard queries and FP queries for some class of queries over "local" models. In the following, we establish connections between relational and XML settings, exploring whether this yields any connections between these complexity results.

4 From Relations to XML

We explain here how to encode probabilistic relational data models into probabilistic XML.

Encoding instances. Given a relational schema $\Sigma = \{(R_i(A_j^i))\}$, we will define the node labels $\langle\top\rangle$, $\langle R_i\rangle$, $\langle A_j^i\rangle$, along with labels representing all possible constants as text values. The root label of XML documents will always be $\langle\top\rangle$. XML representations of instances of the schema will obey the following DTD:

$$\langle\top\rangle : (\langle R_1\rangle^*, ..., \langle R_n\rangle^*)$$
$$\forall i,\ \langle R_i\rangle : (\langle A_1^i\rangle, ..., \langle A_{n_i}^i\rangle)$$
$$\forall i, j,\ \langle A_j^i\rangle : \#\text{PCDATA}$$

We now define the encoding $\langle D\rangle$ of an instance D of Σ. The encoding $\langle R_i(a_1, ..., a_{n_i})\rangle$ of the fact $R_i(a_1, ..., a_{n_i})$ is the subtree whose root has label $\langle R_i\rangle$ and children $\langle A_j^i\rangle$, each child $\langle A_j^i\rangle$ having as child one text node with a label $\langle a_j\rangle$ representing the corresponding a_j. The encoding $\langle D\rangle$ of a full instance D is the XML document whose root has one child $\langle R_i(a_1, ..., a_{n_i})\rangle$ per fact $R_i(a_1, ..., a_{n_i}) \in D$.

Encoding probabilistic instances. We will now define the encoding $\langle \widehat{D} \rangle$ of probabilistic instances $\widehat{D}$ for the various probabilistic relational data models that we described in Section 2.1. The encodings will be given in the probabilistic XML data models of Section 2.2. When we say that we "encode" one probabilistic data model into another, we mean that the following property holds:

$$\forall \widehat{D}, D, \Pr_{\langle \widehat{D} \rangle}(\langle D \rangle) = \Pr_{\widehat{D}}(D) \tag{1}$$

Proposition 1. *Any tuple-independent database can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ *probabilistic document.*

Proof. Given a tuple-independent probabilistic instance $\widehat{D}$, we encode it as a $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ document $\langle \widehat{D} \rangle$ whose root has an ind node as its only child. We root as children of this node the subtrees encoding each of the tuples of $\widehat{D}$, where the probability of the tuple is indicated on the edge attaching it to the ind node. It is easy to see that Equation (1) holds with this encoding. □

Proposition 2. *Any BID can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *probabilistic document.*

Proof. Consider a BID probabilistic $\widehat{\Sigma}$-instance $\widehat{D}$. We will first define an encoding for the blocks of $\widehat{D}$, before defining the encoding $\langle \widehat{D} \rangle$ of $\widehat{D}$.

For all $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$, for all $\mathbf{k} \in \mathbf{K}$ such that $R(\mathbf{k}, \mathbf{a}, p) \in \widehat{D}$ for some $(\mathbf{a}, p) \in \mathbf{A} \times P$, we first define $\langle R(\mathbf{k}, \_, \_) \rangle$ as the subtree whose root has label $\langle R \rangle$, has $|\mathbf{K}|$ children $\langle K_j \rangle$ whose children are text nodes $\langle k_j \rangle$ representing the associated k_j, and has as $(|\mathbf{K}|+1)$-th child a mux node; as children of this mux node, we put one ind node per $\mathbf{a} \in \mathbf{A}$ such that $p = \Pr_{\widehat{D}}(R(\mathbf{k}, \mathbf{a}))$ is > 0, with p as edge label. As children of each of these ind nodes, with edge probability 1, we put $|\mathbf{A}|$ nodes $\langle A_j \rangle$ with children text nodes $\langle a_j \rangle$.

Hence, for each $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$, for each choice of $\mathbf{k} \in \mathbf{K}$, the mux node will select one of the possible choices of $\mathbf{a} \in \mathbf{A}$ based on their probability in $\widehat{D}$, those choices being independent between all of the mux nodes.

As expected, we define $\langle \widehat{D} \rangle$ to be the $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ document whose root has one child $\langle R(\mathbf{k}, \_, \_) \rangle$ per possible choice of $R(\mathbf{K}, \mathbf{A}, P) \in \widehat{\Sigma}$ and $R(\mathbf{k}, \_, \_) \in \widehat{D}$. □

This construction is illustrated in Fig. 3, which is a $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ encoding of the BID database in Fig. 1.

Proposition 3. *Any pc-table can be encoded in linear time as a* $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *probabilistic document with the Boolean formulae unchanged.*

Proof. Consider a probabilistic $\widehat{\Sigma}$-instance $\widehat{D}$ in the pc-table formalism. We will set $\langle \mathcal{V} \rangle$ (the variable set for the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ formalism) to be the variable set $\mathcal{V}$ of $\widehat{D}$, and will simply take $\langle P \rangle(\langle x \rangle)$ to be $\Pr_{\widehat{D}}(x)$.

We now define $\langle \widehat{D} \rangle$ as the $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ document whose root has a fie node as its only child. This node has as descendants the subtrees encoding each of the tuples of $\widehat{D}$, where the condition of the tuple is indicated on the edge attaching it to the fie node. □

Encoding queries. We will now show how a query q on a relational instance D is encoded as a query $\langle q \rangle$ on its XML encoding $\langle D \rangle$. Of course, we will ensure that queries commute with encodings, i.e., the following property holds:

$$\forall \widehat{D}, q,\ q\left(\widehat{D}\right) = \langle q \rangle \left(\left\langle \widehat{D} \right\rangle\right) \tag{2}$$

Proposition 4. *A CQ can be encoded as a TPQJ in linear time.*

Proof. The encoding $\langle a \rangle$ of a constant a is its textual representation, and the encoding $\langle x \rangle$ of a variable x of the query is a TPQJ variable. The encoding $\langle F \rangle$ of an atomic formula $F = R(z_1, ..., z_n)$ over the relation $R(A_1, ..., A_n)$ is the subtree whose root has label $\langle R \rangle$ and has n children; each child has label $\langle A_i \rangle$ and has one child $\langle z_i \rangle$. The encoding $\langle q \rangle$ of a CQ q is a tree whose $\langle \top \rangle$-labeled root has one child $\langle F \rangle$ per atomic formula F in q. □

This encoding is shown in Fig. 2, a TPQJ that encodes q_{Boston}. We rediscover with this result the $\mathsf{FP}^{\#\mathsf{P}}$ membership of CQ evaluation over pc-tables given that of TPQJ over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$. We also obtain the $\mathsf{FP}^{\#\mathsf{P}}$-hardness of TPQJ over $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ given that of CQs over tuple-independent databases. We can finally use this result to find individual hard TPQJ queries as those that are encodings of hard CQs over tuple-independent databases (e.g., non-hierarchical CQs).

Proposition 5. *A query in the relational calculus can be encoded as an MSOJ query in linear time; the resulting query does not have any second-order quantifier, i.e., it is actually a first-order query with joins.*

Proof. Let q be a relational calculus query. We denote by V and C respectively the set of variables and constants appearing in q. For each variable x in V, and each occurrence of x in q, we introduce a new variable x_i. We encode subgoals $R(z_1, ..., z_n)$ for the relation $R(A_1, ..., A_n)$ by the following formula: $\exists y \, \langle R \rangle(y) \wedge \bigwedge_j \exists w \, (y \to w \wedge \langle A_j \rangle(w) \wedge w \to \langle z_j \rangle)$ where the encoding $\langle z_i \rangle$ of a constant a is a fresh variable c_a and the encoding of a variable x is the encoding $\langle x_i \rangle$ of the fresh variable for this specific occurrence of x. Let $\langle q \rangle'$ be the MSO formula obtained from q by encoding all of its subgoals in this way. The MSOJ query $\langle q \rangle$ is $\langle q \rangle' \wedge \left(\bigwedge_{x \in V} \exists x \bigwedge_i x \sim x_i\right) \wedge \left(\bigwedge_{c_a} a(c_a)\right)$ where all x_i's and c_a's are existentially quantified at the top level. □

As an immediate consequence of this encoding, if the original first-order query is *read-once* (no query variable is used twice), it is linear-time to evaluate it over BIDs thanks to the linear-time evaluation of MSO over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$. Read-once queries are of limited interest, however.

5 From XML to Relations

We now show, in the reverse direction, how to encode probabilistic XML instances into probabilistic relational models. This problem has been explored in [6] with two solutions proposed:

Schema-based: adapted inlining Assume we have a DTD for possible XML documents. This method transforms a $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic document (but it is easy to adapt the technique to $\mathsf{PrXML}^{\{\mathsf{fie}\}}$) into a pc-table whose schema is derived from the DTD, adapting techniques from [22] for storing XML documents into relations. Queries are translated as in [22], which may result in queries involving a fixpoint operator in the case of recursive DTDs.

Schemaless: adapted XPath accelerator In the absence of a schema, Hollander and van Keulen propose to transform $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ probabilistic documents (again, the same would hold with $\mathsf{PrXML}^{\{\mathsf{fie}\}}$) into pc-tables by using a pre/size/level encoding as in MonetDB/XQuery [23]. Queries are then translated into queries with inequality predicates and arithmetic.

These two translations are used to show that probabilistic XML can be queried on top of a probabilistic relational engine, Trio in [6] or MayBMS in [24]. However, we present here an alternative translation that has the advantage of transforming TPQJs into CQs, without any need for recursion or inequality predicates.

Encoding instances. Let d be an XML document. We construct the relational schema $\Sigma = \{\mathit{Label}(id, lab), \mathit{Child}(id, cid), \mathit{Desc}(id, did), \mathit{Root}(id)\}$ and encode each node n by a unique ID $\langle n\rangle$. The encoding $\langle d\rangle$ of d is a relation over Σ defined as follows:

- for every node n of d with label l, we add a fact $\mathit{Label}(\langle n\rangle, \langle l\rangle)$;
- for every edge (n, n') in d, we add a fact $\mathit{Child}(\langle n\rangle, \langle n'\rangle)$;
- for every node n and descendant n' of n in d, we add a fact $\mathit{Desc}(\langle n\rangle, \langle n'\rangle)$;
- we add a single fact $\mathit{Root}(\langle r\rangle)$ for the root r of d.

Note that this construction is quadratic at worst since we may add linearly many *Desc* facts for every node n.

Encoding probabilistic instances. We will now encode a probabilistic XML document $\hat{d}$ from Section 2.2 into a probabilistic relational instance $\langle\hat{d}\rangle$ of Section 2.1. Again, the goal is to have an encoding that satisfies (1).

We start with a negative result that shows that tuple-independent databases or even BIDs are unusable for encoding even simple probabilistic documents:

Proposition 6. *No probabilistic document of* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *with more than one possible world can be encoded as a BID (with the instance encoding above).*

Proof. Assume by way of contradiction we have such a document $\hat{d}$ and its encoding $\langle\hat{d}\rangle$. Since $\hat{d}$ has more than one possible world, it contains at least one ind or mux node, say x, and there is a child y of x such that the edge probability label between x and y is $p < 1$. Let z be the lowest ancestor of x that is neither an ind or mux node; if y is itself an ind or mux node, we take y to be its highest descendant that is neither an ind or mux node.

There exists possible worlds d and d' of $\hat{d}$ such that, in $\langle d\rangle$, $\mathit{Child}(\langle z\rangle, \langle y\rangle)$ and $\mathit{Desc}(\langle z\rangle, \langle y\rangle)$ holds, while in $\langle d'\rangle$, neither of these facts hold. But then, since the *Child* and *Desc* tables are fully independent in a BID, there is a possible world of $\langle\hat{d}\rangle$ where $\mathit{Child}(\langle z\rangle, \langle y\rangle)$ holds and $\mathit{Desc}(\langle z\rangle, \langle y\rangle)$ does not, which is absurd, since no encoding of a possible world of $\hat{d}$ verifies this. □

This result may seem to be an artifact of the particular encoding of instances we chose. However, there is something deeper happening here: BIDs are not able to correlate probabilities of tuples, which means they will not be able in particular to represent *hierarchies of* ind *nodes* [14]. More generally, the main issue comes from the fact that BIDs are not a *strong representation system* [10] for the language of (non-Boolean) conjunctive queries: the output of a conjunctive query over BIDs cannot in general be represented as a BID; on the other hand, $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ can represent any discrete probability distribution, and can therefore represent the output of any query over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$. This mismatch means it is hopeless to come up with an alternative way of encoding instances that would make BIDs sufficient to encode $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$.

On the other hand, pc-tables can encode probabilistic XML documents:

Proposition 7. *Any* $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ *or* $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ *probabilistic document can be encoded as a pc-table in cubic time.*

Proof. We restrict to $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ since $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ can be tractably encoded into $\mathsf{PrXML}^{\{\mathsf{fie}\}}$. We first remove every fie node of the probabilistic document by connecting each non-fie descendant n' of a fie node to its lowest non-fie ancestor n, labeling the edge (n, n') with the conjunction of all formulae appearing as labels on the edges of the original path from n to n'. We then construct a pc-table from this document as if it were a non-probabilistic document, except that to each tuple $\mathit{Child}(\langle n\rangle, \langle n'\rangle)$ we add the Boolean condition that appears as label on (n, n'), and to each tuple $\mathit{Desc}(\langle n\rangle, \langle n'\rangle)$ we add the conjunction of all Boolean conditions that appear as labels on the edges of the path between n and n'. At worst, this results in a cubic construction: for every node, for every descendant of this node, we have a condition that has at most linear size. □

Encoding queries. Again, our goal is an encoding of tree queries that satisfies (2).

Proposition 8. *A TPQJ can be encoded as a CQ in linear time.*

Proof. Let q be a TPQJ. The CQ $\langle q\rangle$ is the conjunction of the following atomic formulae:

- for every node n of q with constant or variable label l, an atom $\mathit{Label}(\langle n\rangle, \langle l\rangle)$;
- for every child edge (n, n'), an atom $\mathit{Child}(\langle n\rangle, \langle n'\rangle)$
- for every descendant edge (n, n'), an atom $\mathit{Desc}(\langle n\rangle, \langle n'\rangle)$. □

This can be used to reestablish the $\mathsf{FP}^{\#\mathsf{P}}$ membership of TPQJ over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ from the similar result over relations, or, for example, the $\mathsf{FP}^{\#\mathsf{P}}$-hardness of any encoding of a TPQJ with a single join. Note that the encoding of any TPQ with depth greater than 1 will be non-hierarchical but still tractable on the encodings of $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$: we cannot just use the fact that non-hierarchical queries are intractable since we are working with a specific class of databases.

MSOJ queries cannot be encoded into the relational calculus, since they can express such things as the existence of a path between nodes of a graph (this graph being represented as a tree, e.g., as in Section 4), which is impossible to test in first-order logic. [25]

6 Conclusion

We have thus established connections between probabilistic relational and XML models, showing how probabilistic instances and queries from one model can be encoded into the other. Though we can rediscover some general results in this way (such as the $\mathsf{FP}^{\#\mathsf{P}}$-completeness of query evaluation), we also see that results over probabilistic XML (such as the linear-time evaluation of MSO queries, or the dichotomy for TPQJ queries with a single join) are not direct translations of similar results in the relational case but deep consequences of the true structure.

To go further, one direction would be to look at the tree-width (of the data structure, of the queries, of the Boolean formulae) as an indicator of the tractability of a query; Jha and Suciu have shown [26] that it is tractable to evaluate the probability of a bounded tree-width Boolean function. It is still an open problem to understand the dependency between the tree-width of a query lineage and the tree-width of the query, of the data and of the Boolean formulae. This could suggest new tractable classes of probabilistic relational databases, inspired by the tractable classes of probabilistic XML.

We have restricted our study to discrete finite probabilistic distributions; models for discrete infinite distributions arise naturally in probabilistic XML [27] by adding probabilities to an XML schema [28]; their meaning is less clear in the relational setting. Probabilistic models with continuous distributions can also be defined in the relational [29] and XML [30] cases, though the precise semantics can be tricky. Moreover, no strong representation systems (say, for conjunctive queries) involving continuous distributions have been put forward yet.

References

1. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)
2. Kimelfeld, B., Senellart, P.: Probabilistic XML: Models and complexity (September 2011) Preprint available at `http://pierre.senellart.com/publications/kimelfeld2012probabilistic.pdf`.
3. Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: SIGMOD. (2009)
4. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: CIDR. (2005)
5. Souihli, A., Senellart, P.: Optimizing approximations of DNF query lineage in probabilistic XML. In: Proc. ICDE. (April 2013)
6. Hollander, E., van Keulen, M.: Storing and querying probabilistic XML using a probabilistic relational DBMS. In: MUD. (2010)
7. Lakshmanan, L.V.S., Leone, N., Ross, R.B., Subrahmanian, V.S.: ProbView: A flexible probabilistic database system. ACM Transactions on Database Systems **22**(3) (1997)
8. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB Journal **16**(4) (2007)
9. Green, T.J., Tannen, V.: Models for incomplete and probabilistic information. In: Proc. EDBT Workshops, IIDB. (March 2006)

10. Sarma, A.D., Benjelloun, O., Halevy, A.Y., Widom, J.: Working models for uncertain data. In: ICDE. (2006)
11. Barbará, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Transactions on Knowledge and Data Engineering **4**(5) (1992)
12. Ré, C., Suciu, D.: Materialized views in probabilistic databases: for information exchange and query optimization. In: VLDB. (2007)
13. Nierman, A., Jagadish, H.V.: ProTDB: Probabilistic data in XML. In: VLDB. (2002)
14. Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic XML models. VLDB Journal **18**(5) (October 2009)
15. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
16. Dalvi, N.N., Suciu, D.: Management of probabilistic data: foundations and challenges. In: PODS. (2007)
17. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science **8** (1979)
18. Dalvi, N.N., Schnaitter, K., Suciu, D.: Computing query probability with incidence algebras. In: PODS. (2010)
19. Kharlamov, E., Nutt, W., Senellart, P.: Value joins are expensive over (probabilistic) XML. In: Proc. LID. (March 2011)
20. Cohen, S., Kimelfeld, B., Sagiv, Y.: Running tree automata on probabilistic XML. In: PODS. (2009)
21. Kimelfeld, B., Kosharovsky, Y., Sagiv, Y.: Query evaluation over probabilistic XML. VLDB Journal **18**(5) (2009)
22. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F.: Relational databases for querying XML documents: Limitations and opportunities. In: VLDB. (1999)
23. Boncz, P.A., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In: SIGMOD. (2006)
24. Stapersma, P.: Efficient query evaluation on probabilistic XML data. Master's thesis, University of Twente (2012)
25. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
26. Jha, A.K., Suciu, D.: On the tractability of query compilation and bounded treewidth. In: ICDT. (2012)
27. Benedikt, M., Kharlamov, E., Olteanu, D., Senellart, P.: Probabilistic XML via Markov chains. Proceedings of the VLDB Endowment **3**(1) (September 2010)
28. Abiteboul, S., Amsterdamer, Y., Deutch, D., Milo, T., Senellart, P.: Finding optimal probabilistic generators for XML collections. In: Proc. ICDT. (March 2012)
29. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD. (2003)
30. Abiteboul, S., Chan, T.H.H., Kharlamov, E., Nutt, W., Senellart, P.: Capturing continuous data and answering aggregate queries in probabilistic XML. ACM Transactions on Database Systems **36**(4) (2011)